```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define N 5

sem_t forks[N];
sem_t mutex;

void* philosopher(void* num) {
    int id = *(int*)num;

    while (1) {
        printf("Philosopher %d is thinking...\n", id);
        sleep(1);

        // Try to pick up forks
        sem_wait(&mutex); // Critical section to avoid deadlock

        sem_wait(&forks[id]);             // Pick up left fork
        sem_wait(&forks[(id + 1) % N]);   // Pick up right fork

        sem_post(&mutex);

        // Eating
        printf("Philosopher %d is eating 🍝\n", id);
```

```
        sleep(2);

        // Put down forks
        sem_post(&forks[id]);
        sem_post(&forks[(id + 1) % N]);

        printf("Philosopher %d put down forks and is thinking again.\n", id);
    }
}

int main() {
    pthread_t thread_id[N];
    int i, ids[N];

    // Initialize semaphores
    for (i = 0; i < N; i++)
        sem_init(&forks[i], 0, 1); // each fork is available

    sem_init(&mutex, 0, 1); // to control access to critical section

    // Create philosopher threads
    for (i = 0; i < N; i++) {
        ids[i] = i;
        pthread_create(&thread_id[i], NULL, philosopher, &ids[i]);
    }

    // Join threads (infinite loop, so not necessary, but for completeness)
    for (i = 0; i < N; i++)
```

```
        pthread_join(thread_id[i], NULL);


    return 0;

}
```

```
Philosopher 0 is thinking...
Philosopher 1 is thinking...
Philosopher 2 is thinking...
Philosopher 3 is thinking...
Philosopher 4 is thinking...
Philosopher 0 is eating 🍒
Philosopher 0 put down forks and is thinking again.
Philosopher 0 is thinking...
Philosopher 1 is eating 🍒
Philosopher 1 put down forks and is thinking again.
Philosopher 2 is eating 🍒
Philosopher 1 is thinking...
Philosopher 2 put down forks and is thinking again.
Philosopher 2 is thinking...
Philosopher 3 is eating 🍒
```